NTDR INFOSEC: SOFTWARE SECURITY AND THE 80486 MICROPROCESSOR

S. Kranjac McIntosh

ITT Industries Aerospace/Communications Division
Secure Information Systems
Clifton, New Jersey

1. ABSTRACT

This paper discusses one of the first efforts of developing a software-based secure communication system that bases security on a combination of hardware and software design features. The paper starts with an overview of the security relevant features of the 80486 architecture and draws a contrast between the protected and real mode environments of the 80486 microprocessor. In particular, protected mode features including segments, descriptor tables, gates, privilege levels and Task State Segments are presented as a prelude to a more detailed discussion of each feature later in the paper. The value of each of these features in the development of a trusted operating environment is explored. This groundwork in place, the paper describes the application of the 80486 features to the development of security software for the Mercury NTDR (Near Term Digital Radio). The detailed description of the software architecture closes with a discussion of the role of the 80486 features in software assurance.

2. PAPER OVERVIEW

The purpose of the Mercury NTDR Information Security (Infosec) Subsystem is to provide the Mercury NTDR system with encrypt/decrypt capabilities to secure communications over a wireless network.

This paper describes the role of the 80486 microprocessor in assuring system security and data integrity in the context of the NTDR Infosec software design. Of particular interest are the protected mode features of the 80486 microprocessor.

The multi-segmented memory model of the 80486 protected mode facilitates the required separation of red data from black data in the system. Since the red and black data are not physically separated, it is desirable to have a guard watching the borders. Ideally, the guard is on duty at all times, can detect and report violations and operates independently of the hosted software. The 80486 satisfies these requirements.

When run in protected mode, the 80486 can allow or deny memory accesses in real-time by processing customized rules that are independent of the 'program' that is running. Microcode inherent to the 80486 monitors the memory accesses at a fine level of granularity. Customized access rules linked into the executable image are the security backbone of the system and serve to de-couple spheres of processing.

The value of each of the 80486 protected mode features employed in the NTDR Infosec design is discussed in greater detail in the sections which follow. The capability of the 80486 to detect and flag failures due to faulty components or erroneous code in protected mode is also explored.

3. 80486 PROTECTED MODE BUILDING BLOCKS

This paper covers the development of a secure operating system hosted on an 80486. Although not all portions of an operating system are trusted, the design philosophy is: *strong fences make good neighbors*. To this end, we enlist the protected mode features of the 80486 microprocessor. This mode offers the designer many tools useful in the building of secure software.

The 80486 16-bit real mode is the default mode of the 80486 microprocessor following reset. When the processor operates in real mode, control transfers and memory accesses occur without constraint, except for minimal address range checking. The protected mode, by contrast, limits control transfers, performs automatic privilege level checks on control transfers and data accesses and enforces both locally and globally defined access rights all on a task by task basis.

How does the 80486 know which operations to allow and which to deny? There are several protected mode system structures which capture the 'rules' of the system. These rules are formed by the operating system designer during the design phase and are then allocated to the appropriate system structures. The structures are linked into the final executable image. At run time, some of these structures are necessarily copied to their final destinations in RAM where they may be written to by the 80486 microcode as the system runs.

In protected mode, the basic building block is the segment. A segment delineates a contiguous block of bytes with common attributes.

0-7803-6521-6/$10.00 (C) 2000 IEEE

A segment descriptor is another building block. It is comprised of eight bytes of encoded information about a segment. A descriptor is a receptacle used to hold a segment's attributes. In just eight bytes, a descriptor can fully capture the attributes of a contiguous block of memory. This is the information that the 80486 uses to validate privileges, write permission, etc. **at run time**. The descriptors for segments that are common to the whole system can be gathered into a descriptor table called the system Global Descriptor Table, or GDT. The descriptors for segments of memory that should not be widely available are allocated to one or more Local Descriptor Tables, or LDTs.

Finally, we have selectors. A selector selects a descriptor in either the GDT or the current LDT. Where does the selector come from? When a routine is compiled and linked, the memory accesses specified in the instructions resolve to pairs written as *selector:offset*. The role of the offset is to pinpoint an exact byte some distance from the start of the segment base address.

3.1 SEGMENTS

A *segment* is a logical grouping of one or more contiguous bytes of physical memory which share common attributes. The attributes that are specified for a segment, such as base address, limit, and access rights are captured in a receptacle called a *descriptor*.

There are several categories of segments. Code segments typically reside in read-only memory and contain the executable code of a system as well as any ROM-able tables. Data segments typically reside in RAM. Stack segments always reside in RAM. Task State Segments (TSSs) reside in RAM and hold task context information.

3.2 TABLES

A segment is described by a descriptor. The system designer assigns each descriptor to the Global Descriptor Table or to one or more Local Descriptor Tables.

An Interrupt Descriptor Table (IDT) contains descriptors that are used in handling interrupts. These descriptors can be task gate descriptors, trap gate descriptors and interrupt gate descriptors.

3.3 DESCRIPTORS

A descriptor provides detailed information about a memory segment or other 80486 protected mode construct.

For example, there are descriptors for code segments, data segments, stack segments, TSSs, and LDTs. [1, 2, 3] The 80486 protected mode architecture also provides for interrupt gate, trap gate, call gate, and task gate descriptors.

3.4 SELECTORS

Selectors are used to select a table and a descriptor within that table whether the table be the GDT or an LDT. [3] The three fields of a selector are: the Table Index, the Table Indicator and the Requestor Privilege Level. The Table Index is used to index into the table indicated by the Table Indicator. If the Table Indicator equals zero, the GDT is the target. Otherwise, the current LDT is the target table.

4. 80486 PROTECTED MODE IN PRACTICE

The study of the 80486 protected mode features can be a daunting task as the concepts are so interdependent. To understand selectors, one must study descriptors which in turn demand an understanding of segments and tables. In the previous section, some introductory material was presented in an effort to introduce the reader to protected mode concepts and terminology.

4.1 USING THE GLOBAL DESCRIPTOR TABLE

The GDT can reside in read-only memory, however, for systems that implement the 80486 tasking model the GDT must be writeable because the 80486 updates and verifies the TSS descriptors during task switching.

4.2 USING THE LOCAL DESCRIPTOR TABLE

LDTs can be safely linked to ROM locations in the executable as long as the operating system does not track the 'accessed bit'. When code, data, and stack segment are accessed, the 80486 sets the 'accessed bit' in the segment descriptor. [3] Some operating systems manage memory resources by monitoring the accessed bit. For those types of systems, the LDTs need to be allocated to writeable memory.

4.3 CODE SEGMENT SELECTION

As a program runs, the 80486 performs validation of code, data, and stack segments being accessed by carefully analyzing the segment descriptor content. [3] For code segments, there can be a privilege check, offset limit validation, access rights validation and selector validation. As an example, consider the 80486 JMP instruction. Before performing the jump, the 80486 performs several checks.

[1, 2, 3] For *every* intra-segment jump (near jump) the offset of the destination must be within the limits of the current segment.

[1, 2, 3] For *every* inter-segment jump (far jump), the given selector is scrutinized by the 80486. The selector must not be null. The table indicator must identify a valid table. The table index must not exceed the limit of the table specified.

[3] Next, the descriptor selected is scrutinized by the 80486. The Segment Present bit must be set. The privilege level check is performed. The offset portion of the selector:offset pair must be within the code segment limit.

The inter-segment checks are performed at every task switch resulting from a call of another task, a return to a task, the servicing of an interrupt by a task or a simple inter-segment jump or call. These checks are supplemented by checks at the task gate, TSS, data segment and stack segment levels to be discussed in later sections.

The selector checks are performed when the large memory model is used and there is a memory access attempt. The selector checks also occur when certain 80486 registers, known as segment registers, are loaded with selectors in the course of program execution.

4.4 DATA SEGMENT SELECTION

[1, 2, 3] Prior to access of a data segment, a privilege check is performed, the offset is validated, access rights are validated and the selector is validated. These checks are performed automatically by the 80486.

[3] For *every* intra-segment access (near access) the offset of the destination must be within the limits of the target segment.

[3] For *every* inter-segment access, (far access), the given selector is scrutinized by the 80486 prior to being loaded into one of the data segment registers. The selector should not be null. If it is, no error is generated until the selector is used. The table indicator is validated. The table index provided by the selector must not exceed the limit of the table specified.

[3] Next, the descriptor selected by the selector is scrutinized by the 80486. The Segment Present bit must be set. The privilege level check is performed. The Access Rights byte must be valid. The offset must be within the data segment limit.

4.5 STACK SEGMENT SELECTION

[3] Before a stack segment is accessed its descriptor is validated. For stack segments, the descriptor must indicate that the target memory can be written to. In addition, the privilege is checked and the offset limit, access rights and selector are validated. For example, consider the 80486 MOV and PUSH instruction. Before performing the PUSH, the 80486 performs a list of checks. For *every* intra-segment access (near access) the new value of the stack pointer must be valid. The target stack segment must be writeable.

[3] For *every* inter-segment access (far access) the given selector is scrutinized by the 80486 prior to a MOV instruction loading the SS register with the selector. The selector must not be null. The table indicator must be valid. The table index must not exceed the limit of the table specified.

[3] Next, the descriptor selected by the selector is scrutinized by the 80486 during a PUSH instruction. The Segment Present bit must be set. The RPL, the CPL and the DPL must be equal. The target stack segment must be writeable. The new value of the stack pointer must be valid.

The selector checks are performed when the Stack Segment (SS) register is loaded with a new selector value.

4.6 TASK STATE SEGMENT SELECTION

The 80486 performs validation of TSSs during task switches into and out of tasks. [3] Prior to *every* task switch operation, the TSS selector is scrutinized by the 80486. The selector must not be null. The table indicator must select the GDT and the GDT must be installed and valid. The table index must not exceed the limit of the GDT.

Next, the TSS descriptor selected by the TSS selector is scrutinized by the 80486. For a task call, the TSS descriptor Busy bit must indicate that the task is idle. For a return, using the IRET instruction, the Busy bit of the task must indicate that the task's last status was Busy. The Segment Present bit must be set. A privilege level check is performed.

Finally, the 80486 validates the data contained in the target TSS as it prepares to load the context of the TSS. This includes validation of the following selectors and descriptors: LDT, code segment, data segment, stack segment.

4.7 TASK GATE SELECTION

Just as code, data, stack and TSSs are validated as a program runs, the 80486 performs validation of task gates during task switches into tasks. Note that unlike the other segments that we have discussed thus far, a task gate descriptor does not reference a 'task gate segment'. Instead, a task gate references a TSS via a TSS selector that is stored in the task gate descriptor.

[3] Prior to *every* task switch operation through a task gate, the given task gate selector is scrutinized by the 80486. The table indicated by the table indicator must be installed and valid. The table index provided by the selector must

not exceed the limit of the table. The task gate descriptor must be valid. The TSS selector must identify a valid TSS.

Next, the TSS selector and descriptor are scrutinized by the 80486 as was covered in section 4.6, Task State Segment Selection.

Finally, the 80486 validates the data contained in the target TSS as it prepares to load the context of the target TSS.

These checks are performed on every task switch in the system. The Infosec design uses task gates for entry into every task in the system, including the interrupt handlers.

4.8 LOCAL DESCRIPTOR TABLE SELECTION

Before an LDT is activated, the LDT selector and the LDT descriptor are validated by the 80486. In the Infosec system, all LDTs are linked into ROM locations and remain there during system operation.

[3] Prior to *every* task switch operation, the selector of the LDT to be loaded is scrutinized by the 80486. The table indicator must select the GDT and the GDT must be installed and valid. The table index must be within the limits of the GDT. The LDT descriptor must be valid.

5. INFOSEC 80486 PROTECTED MODE FEATURES

This section details all 80486 features employed in the design of the NTDR Infosec software. We will draw on the material discussed in previous sections as the Infosec design is analyzed.

5.1 INFOSEC 80486 TASKING

The NTDR Infosec software is partitioned into software security groups. Each software security group is comprised of one or more 80486 tasks. A task is defined to the 80486 using a TSS, a system structure.

Each Infosec task is defined by its attributes which are held by the task's TSS. There are many attributes but the key ones define the task's code segment (CS), data segment (DS), stack segment (SS) and LDT.

In addition to attributes, a TSS is also the receptacle for the context of the task. When a task is called, the 80486 stores the context of the current task, the caller, to the current task's TSS. The 80486 then attempts to load the callee's TSS. At this point, the 80486 validates the data in the callee's TSS. If any of these validation checks fail, the 80486 processor raises an exception and the Infosec shuts down. If all checks are successful, the 80486 loads its CPU registers with the data found in the callee's TSS. [2] The data to be loaded includes the CS, DS, SS, EIP, LDTR, and EFLAGS CPU registers.

In addition, a backlink to the caller is stored in the callee's TSS to be used in return sequence. If the 80486 determines that the backlink field contains invalid data, it raises an exception and the Infosec shuts down. If the backlink field is valid, as it should be, the context of the caller identified by the backlink field is restored. Once again, prior to loading the 80486 registers with the data last stored in the caller's TSS, the 80486 validates the data.

The design of the interrupt handling features of the Infosec is modeled after the caller/callee sequence described above. The interrupted task has an opportunity to store its context before the interrupt handler task is loaded. Upon return from interrupt, the context of the interrupted task is reloaded and execution continues seamlessly.

5.1.1 NON-MASKABLE INTERRUPT HANDLER TASKS

Non-maskable interrupt service routines in the Infosec implementation culminate in either the shutdown or reset of the processor. The first NMI which occurs is serviced at the very next instruction boundary when the 80486 vectors to the appropriate handler.

The Alarm Handler Task handles all alarms and zeroize requests and runs at privilege level 0. This task is invoked when interrupt 2, the 80486 Non-Maskable Interrupt, is generated by the FPGA.

The Exception Handler Task handles all internal 80486 exceptions. The Exception Handler task runs at privilege level 0.

5.1.2 MASKABLE INTERRUPT HANDLER TASKS

The 1 Millisecond Timer Interrupt Handler Task responds to the one millisecond timer interrupt generated by Infosec hardware and maintains and checks critical timers. This task runs at privilege level 0.

The Protocol Processor Interrupt Handler Task responds to buffer and soft reset interrupts from the Protocol Processor. This task runs at privilege level 0.

The Waveform Processor Interrupt Handler Task responds to buffer interrupts from the Waveform Processor. This task runs at privilege level 0.

The DS101 Controller Interrupt Handler Task responds to data and status interrupts from the DS101 controller chip. The DS101 Controller Interrupt Handler task runs at privilege level 0.

5.1.3 INFOSEC APPLICATION TASKS

There are a number of Infosec application tasks. These tasks run at various 80486 privilege levels consistent with

the security group guidelines specified for the Infosec system.

The Scheduler Task runs at privilege level 1 and is responsible for the dispatch of other Infosec tasks in response to state, substate and error code information.

The Initialization Task is called during the Infosec power up sequence to initialize the system and perform power up tests.

The Find Work Task is called by the Scheduler task when the system is idle. Task Find Work identifies and prioritizes pending processing requests. It runs at privilege level 0 to facilitate the scheduling process.

The Background Built In Test Task is called periodically by the Scheduler task. The BgBIT task is responsible for testing an increment of RAM or ROM with each call.

The Initiated/Periodic Built In Test Task is dispatched by the Scheduler task when the FindWork task has detected a 'Run BIT' message from the Protocol Processor.

The Find Message Task serves as a Scheduler utility. It runs at privilege level 0 to facilitate and expedite message processing.

The Fill Task runs at privilege level 2 and performs key loading operations.

The Untrusted Red Tasks run at privilege level 3 and perform processing on the red side (plaintext) of the Infosec board.

The Untrusted Black Tasks run at privilege level 3 and perform processing on the black side (cyphertext) of the Infosec board.

The Trusted Bypass Task runs at privilege level 2. It is trusted because it has access to both the red and black sides of the Infosec.

5.2 LOCAL DESCRIPTOR TABLES

The role of an LDT is to fence in a task. The LDT delineates exactly which segments are available to a task and to what extent. As the name implies, an LDT is a table which holds descriptors. As a safeguard, the Infosec system keeps all LDTs in read-only memory.

5.3 INFOSEC GLOBAL DESCRIPTOR TABLE

A GDT delineates exactly which segments are available to all system tasks, and to what extent. As the name implies, a GDT is a table in memory that holds global descriptors. The Infosec GDT must reside in writeable memory because context information, particularly for TSS descriptors, is updated and validated by the 80486 automatically upon call and return of tasks.

5.4 INFOSEC AND 80486 PRIVILEGE LEVELS

[1, 2, 3] The protected mode provides for four privilege levels, 0 being the highest and 3 being the lowest. How is the CPL determined when running in protected mode? For most systems, the privilege level of the code segment which contains the code that is currently executing is the CPL. The 80486 uses the CPL to grant or deny the currently executing code access to other segments.

6. 80486 PROTECTED MODE FEATURES AND INFOSEC SOFTWARE ASSURANCE

The Infosec system structures, including the GDT, IDT, LDTs, TSSs, code, data, and stack segment descriptors, and task gate descriptors together with deliberate privilege level allocation, access allocation, ROM vs. RAM placement, fixed stack and data segment sizes and information hiding provide for built-in layers of protection. These structures, as implemented in the Infosec software design, provide multiple levels of real-time software flow verification as well as real-time data flow verification - all at runtime. These structures are introduced into the system at the link phase and are present and utilized at runtime. The data that these structures provide to the 80486 undergo a variety of data integrity checks with every access of the data. If any data fails the integrity check, the 80486 raises an appropriate exception and the Infosec software shuts down the system.

In this way, attempts by rogue or corrupted software to access data illegally can be detected at runtime by the 80486. In addition, component failures such as ROM bit errors and RAM bit errors in specific GDT, IDT, LDT and descriptor fields can be detected by the 80486 at runtime and flagged to the Infosec software.

6.1 TASK GATES AND SECURITY

The Infosec software design utilizes task gates as an added measure for strengthening the security perimeter of a task. All task gates in the Infosec system are defined in LDTs to limit their availability.

In clarifying the need for task gates, it is useful to briefly discuss TSS descriptors. [2] In the Infosec system, all TSS descriptors have a DPL of 0 and are stored in the GDT as required. Since the all descriptors in the GDT are available to all tasks, one might expect that any task can call any other task. [3] This is not the case. According to the rules of privilege, a TSS with privilege level 0 can only be called by another task that is running at privilege level 0. In the Infosec system, many tasks run at lower privilege

levels as identified by their CPL. To enable a less privileged task to pass control to a more privileged task, a task gate is used. [3] The TSS privilege level check is replaced by a privilege level check at the task gate, the first line of defense. Each task gate DPL is designed to match the anticipated caller's privilege level (CPL) to ensure a successful call.

6.2 THE TSS AND SECURITY

6.2.1 THE ROLE OF THE TSS SELECTOR

The next level of defense from undetected component failure in the Infosec system is supplied by a task's TSS selector. A selector that references a TSS descriptor always points to the GDT because TSS descriptors can only be defined in the GDT. As an example, if the TSS selector specifies that the TSS descriptor is in the LDT, the 80486 raises an exception and the Infosec shuts the system down since this is an illegal selector. An invalid selector could be the result of a ROM component failure which has corrupted the TSS selector. The index field of the TSS selector is also validated by the 80486. An invalid index causes the 80486 to generate an exception, and the Infosec shuts down.

6.2.2 THE ROLE OF THE TSS DESCRIPTOR

A task's TSS descriptor provides the next level of defense from undetected component failure in the Infosec system.

In performing a task switch, the 80486 performs several validation steps. One worth mentioning is the TSS DPL check. This is of interest because we know that a TSS descriptor must be allocated to the GDT, thus making the descriptor available to all tasks in the system. [1, 2, 3] In order for a task to successfully access the target TSS, the task must be at least as privileged as the targeted TSS. If a non kernel-level task should erroneously attempt to access a TSS directly, without passing through a task gate, an Invalid TSS exception is raised by the 80486 and the Infosec shuts down.

As the Infosec software executes, task switches occur as a result of calls, interrupts and returns. [3] When a task switch is initiated for either a call or a return, the Busy Bit in the callee's TSS descriptor is validated prior to being written to by the 80486. If the TSS descriptor is already marked busy when a call is attempted, the 80486 raises an exception and the Infosec shuts down. [3] If the TSS descriptor is already marked idle on a return, the 80486 raises an exception and the Infosec shuts down. These checks are automatically performed by the 80486 and cannot be disabled while in protected mode. An incorrect value stored at the Busy Bit of the TSS descriptor could be a sign of RAM component failure or the result of an erroneous write to RAM.

In addition to the Busy Bit, the TSS descriptor also contains other fields that are validated by the 80486 upon each task switch.

The task switch processing for a return from a callee task to the caller task is via the backlink field in the callee's TSS. The backlink field contains the selector that points to the caller's TSS. The return task switch is performed directly via the TSS descriptor, using the backlink that was stored at the time of the call. [2] As with all selectors, the 80486 performs the prescribed validation checks for an IRET, return from interrupt, instruction on all task returns. The selector stored in the backlink field must point to a valid TSS descriptor that is within the boundaries of the GDT. The TSS descriptor must indicate that the task to be returned to is marked Busy. Any failure in the validation of the selector causes the 80486 to raise an exception. In response, the Infosec shuts the system down.

6.2.3 THE ROLE OF THE TSS

Finally, a task's TSS supplies valuable information that can enable the 80486 to detect a component failure. A TSS indicates to the 80486 where the task's code, data, stack and LDT can be found.

If the Infosec software attempts to write to a TSS, the 80486 raises an exception and the Infosec shuts the system down. [1, 2] The 80486 detects the invalid access because the 'System' bit of a TSS descriptor is set to zero to identify the TSS as a system structure. The 80486 prevents direct access to such structures.

Once the TSS selector and descriptor have passed the 80486 validation check, the TSS attribute selectors, (for code, data, stack and LDT), are checked.

6.3 THE GDT AND SECURITY

The 80486 prevents the Infosec software from directly accessing the GDT and all other system structures. If the Infosec software attempt to write to the GDT directly, the 80486 raises an exception and the Infosec shuts down.

6.4 THE LDT AND SECURITY

6.4.1 THE ROLE OF THE LDT SELECTOR

[2] A selector that references an LDT descriptor always points to the GDT because LDT descriptors can only be defined in the GDT. If the LDT selector specifies that the LDT descriptor is in an LDT, the 80486 raises an exception and the Infosec shuts down. An invalid selector could be the result of a RAM bit error or an erroneous write into the TSS which has corrupted the LDT selector. The index field of the LDT selector is also validated -- it must fall within the limits of the GDT.

6.4.2 THE ROLE OF THE LDT DESCRIPTOR

The LDT descriptor is comprised of the following fields which are validated by the 80486: segment present bit, DPL, system bit. A failure in the validation of these fields causes the 80486 to raise an exception and the Infosec software shuts the system down. The LDT descriptor also contains the LDT base address and size.

6.4.3 THE ROLE OF THE LDT

The LDT is a table of descriptors for memory segments, call gates and task gates local to a given task. Only one LDT is active at any given time. [2] A new LDT can replace the current one via a task switch due to a task call or task return or via the Load LDT instruction which is a privileged instruction. [2] If an unprivileged task attempts to issue the LLDT command, the 80486 generates an exception and the Infosec shuts down. In this way, erroneous loading of an LDT by lesser privileged tasks is prevented.

6.5 SELECTORS AND SECURITY

[1, 2, 3] The next step is to validate the selectors with respect to the selector index and RPL. If the index field of a selector is outside of the table boundary, the selector fails validation and causes the 80486 to raise an exception and shut down. Such a failure could be an indication of a RAM component failure, an erroneous write to RAM, or a ROM component failure. A failed privilege level check also results in system shutdown. This type of failure can be attributed to a RAM or ROM failure.

6.6 CODE SEGMENTS AND SECURITY

Protected mode far calls and returns cause the 80486 to attempt to load the Code Segment register with a new selector. [2] First, the selector is validated and then the offset. Finally, the privilege check occurs. Execution proceeds only if all of these tests pass.

6.7 DATA SEGMENTS AND SECURITY

[2] Just as with code segment selectors, the 80486 validates data segment selectors prior to instruction execution. The selector and offset are validated. Finally, the privilege check occurs. If all of these tests pass, the data segment register is loaded with the selector. A failure causes the 80486 to raise an exception and the Infosec shuts the system down.

6.8 STACK SEGMENTS AND SECURITY

[3] Every time a new stack reference occurs, the 80486 validates the address before loading the selector portion of the address into the Stack Segment register. Any access of the stack causes the offset portion of the far address to be validated before being loaded into the 80486 Extended Base Pointer/Extended Stack Pointer. Finally, the privilege check occurs. If all of these tests pass, the stack can be accessed successfully. If any of the validation checks fail, the 80486 raises an exception and the Infosec software shuts the system down.

6.9 THE IDT AND SECURITY

The IDT is accessed by the 80486 when maskable interrupts, non-maskable interrupts, exceptions, and software-generated interrupts occur. The Infosec system installs appropriate task gate descriptors at all 256 IDT locations to explicitly handle every possible interrupt. The protected mode IDT is installed in ROM. This enhances system security by preventing any erroneous writes from destroying the integrity of the IDT data.

7. CONCLUSION

In this paper many of the protected mode features of the 80486 microprocessor were presented and described in detail. This paper draws on the experience gained from building a secure embedded operating system with real-time constraints for an 80486 target. The Infosec software design incorporates many protected mode features including selectors, descriptors, segments, LDTs, GDTs, IDTs, task gates, TSSs and tasking. By utilizing these features, the power of the 80486 microprocessor to detect erroneous or unauthorized memory references was harnessed to produce reliable software from early on in the software life cycle.

8. REFERENCES

[1] AM486DX/DX2 Microprocessor Hardware Ref. Manual, Advanced Micro Devices Inc., Rev. 1, 1993

[2] Intel486 Microprocessors and Related Products, Intel Corporation, 1995

[3] Protected Mode Software Architecture, MindShare Inc., Addison-Wesley Publishing Co., 1996